📘 A Neural Computing Approach to Income Prediction: Implementation and Comparison of MLP and SVM on Adult Dataset

This coursework focuses on building and evaluating machine learning models to predict whether an individual's income exceeds $50,000 per year using the **UCI Adult Income dataset**. The goal is to compare the performance of two supervised learning models: **Multi-Layer Perceptron (MLP)** and **Support Vector Machine (SVM)**.

The workflow includes a complete machine learning pipeline starting from **data preprocessing** and handling class imbalance using **SMOTE**, to training **baseline models**, and then improving them using advanced techniques such as **hyperparameter tuning**, **regularization**, **parallelization**, and **randomized search**.

By applying evaluation metrics including **accuracy**, **confusion matrix**, and **ROC-AUC curve**, this coursework aims to identify the most effective model in terms of both prediction performance and computational efficiency.

🔹 Step 1: Load and Inspect Dataset

In this step, we load the UCI Adult Income dataset and replace any missing value placeholders (e.g., '?') with `NaN`.
We explore the structure of the dataset using basic commands to view its shape, column types, and sample entries.
This helps in understanding the data distribution and identifying missing or inconsistent values before preprocessing.

```
# Load and Inspect Dataset

import pandas as pd
```

```python
import numpy as np

# Load the dataset
df = pd.read_csv('adult.csv')  # Adjust path if needed

# Replace missing value placeholders
df.replace('?', np.nan, inplace=True)

# Display basic information
print("✅ Dataset loaded successfully.\n")
print("📊 Shape of the dataset:", df.shape)
print("\n🔍 First 5 rows:")
print(df.head())

print("\n🧠 Column Info:")
print(df.info())

print("\n❓ Number of missing values per column:")
print(df.isnull().sum())
```

```
✅ Dataset loaded successfully.

📊 Shape of the dataset: (48842, 15)

🔍 First 5 rows:
   age  workclass  fnlwgt     education  educational-num      marital-
status  \
0   25    Private  226802          11th                7       Never-
married
1   38    Private   89814       HS-grad                9  Married-civ-
spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-
spouse
3   44    Private  160323  Some-college               10  Married-civ-
spouse
4   18        NaN  103497  Some-college               10       Never-
married

          occupation relationship   race  gender  capital-gain
capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0
```

```
0
1    Farming-fishing      Husband  White    Male             0
0
2    Protective-serv      Husband  White    Male             0
0
3  Machine-op-inspct      Husband  Black    Male          7688
0
4                NaN    Own-child  White  Female             0
0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

🧠 Column Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        46043 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       46033 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   47985 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

```
None

❓ Number of missing values per column:
age                  0
workclass         2799
fnlwgt               0
education            0
educational-num      0
marital-status       0
occupation        2809
relationship         0
race                 0
gender               0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     857
income               0
dtype: int64
```

📊 Statistical Summary for Numerical Features

This section provides summary statistics (mean, std, min, max, etc.) for all numerical columns in the dataset.
It helps in understanding the central tendency, spread, and potential outliers in the numeric data.

```
# 📊 Statistical Summary for Numerical Features

# Select only numerical columns
numerical_cols = df.select_dtypes(include=['int64',
        'float64']).columns

# Display summary statistics
print("📈 Statistical Summary (Numerical Columns Only):\n")
print(df[numerical_cols].describe().transpose())
```

```
📈 Statistical Summary (Numerical Columns Only):

                 count           mean            std      min
25%  \
age            48842.0      38.643585      13.710510     17.0
```

```
28.0
fnlwgt           48842.0  189664.134597  105604.025423  12285.0
117550.5
educational-num  48842.0      10.078089       2.570973      1.0
9.0
capital-gain     48842.0    1079.067626    7452.019058      0.0
0.0
capital-loss     48842.0      87.502314     403.004552      0.0
0.0
hours-per-week   48842.0      40.422382      12.391444      1.0
40.0

                      50%       75%        max
age                  37.0      48.0       90.0
fnlwgt           178144.5  237642.0  1490400.0
educational-num      10.0      12.0       16.0
capital-gain          0.0       0.0    99999.0
capital-loss          0.0       0.0     4356.0
hours-per-week       40.0      45.0       99.0
```

🔍 Exploratory Data Analysis (EDA)

This section provides a comprehensive analysis of the dataset through visualizations.
It starts with understanding the distribution of the target variable (`income`) and continues with univariate analysis for both numerical and categorical features.
We further explore relationships between features and the target using count plots and boxplots.
Finally, a correlation matrix highlights linear relationships between numerical variables, helping us detect multicollinearity and feature importance.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
sns.set(style="whitegrid")

# ----------------------------
# 1. Target Variable Distribution
# ----------------------------
plt.figure(figsize=(6, 4))
```

```
sns.countplot(x='income', data=df)
plt.title('Target Variable Distribution: Income')
plt.xlabel('Income Class')
plt.ylabel('Count')
plt.show()

# ------------------------------
# 2. Univariate Analysis
# ------------------------------

# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['int64',
        'float64']).columns
categorical_cols = df.select_dtypes(include=
        ['object']).columns.drop('income')

# Numerical Features - Histograms
for col in numerical_cols:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

# Categorical Features - Count Plots
for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(y=col, data=df, order=df[col].value_counts().index)
    plt.title(f'Frequency of {col}')
    plt.xlabel('Count')
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()

# ------------------------------
# 3. Bivariate Analysis (Categorical vs Target)
# ------------------------------
for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=col, hue='income', data=df,
        order=df[col].value_counts().index)
```

```
    plt.title(f'{col} vs Income')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Numerical vs Target - Boxplots
for col in numerical_cols:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='income', y=col, data=df)
    plt.title(f'{col} Distribution by Income')
    plt.xlabel('Income')
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()

# ------------------------------
# 4. Correlation Matrix
# ------------------------------
plt.figure(figsize=(10, 6))
corr_matrix = df[numerical_cols].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

Data Preprocessing

In this section, the dataset is preprocessed to ensure smooth model performance and proper splitting. The following steps are performed:

1. **Handling Missing Values**: Any rows with missing data are dropped to ensure that the models work with complete data.

2. **Encoding Categorical Variables**: The target variable 'income' is encoded into binary values using `LabelEncoder` (1 for income > 50K, 0 for income <= 50K). All other categorical variables are one-hot encoded using `pd.get_dummies` to convert them into numerical features.

3. **Normalizing Numerical Features**: Numerical features are scaled using `StandardScaler` to ensure that all variables have the same scale, which is important for models like SVM.

4. **Data Splitting**: The data is split into training and test sets using `train_test_split` from `sklearn.model_selection`, with 80% of the data used for training and 20% for testing. The split is stratified to maintain the class distribution in both sets.

5. **Class Distribution Check**: Before any resampling techniques, the class distribution is checked to understand the balance between the target classes.

These preprocessing steps are crucial for ensuring smooth model execution and effective comparison between the models.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from imblearn.over_sampling import SMOTE


# 1. Handle Missing Values
df_clean = df.dropna().copy()
print(f"✅ Data cleaned: {df.shape[0] - df_clean.shape[0]} rows
        removed due to missing values.")


# 2. Encode Categorical Variables
# First, encode the target
le_income = LabelEncoder()
df_clean['income'] = le_income.fit_transform(df_clean['income'])  #
        >50K = 1, <=50K = 0


# Encode all other categorical variables using one-hot encoding
df_encoded = pd.get_dummies(df_clean, drop_first=True)


# 3. Normalize Numerical Features
scaler = StandardScaler()
```

```
numerical_cols = df.select_dtypes(include=['int64',
        'float64']).columns
df_encoded[numerical_cols] =
        scaler.fit_transform(df_encoded[numerical_cols])

# 4. Split into Train/Test Sets
X = df_encoded.drop('income', axis=1)
y = df_encoded['income']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"\nTraining samples: {X_train.shape[0]}")
print(f"Test samples: {X_test.shape[0]}")
print(f"Class distribution (before SMOTE):\n{y_train.value_counts()}")
```

✅ Data cleaned: 3620 rows removed due to missing values.

```
Training samples: 36177
Test samples: 9045
Class distribution (before SMOTE):
income
0    27211
1     8966
Name: count, dtype: int64
```

Handling Class Imbalance with SMOTE

To address the class imbalance in the dataset, SMOTE (Synthetic Minority Over-sampling Technique) is applied to the training data. Before SMOTE, the class distribution shows a significant imbalance, with 27,211 instances of the majority class (income <= 50K) and 8,966 instances of the minority class (income > 50K). After applying SMOTE, synthetic samples are generated for the minority class, balancing the classes. This step helps improve model performance, especially for algorithms sensitive to class imbalance like SVM. The class distribution is now equalized, as shown by the post-SMOTE counts.

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to balance the classes
smote = SMOTE(random_state=42)
```

```
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

print("Before SMOTE:", np.bincount(y_train))
print("After SMOTE:", np.bincount(y_resampled))
```

```
Before SMOTE: [27211  8966]
After SMOTE: [27211 27211]
```

```
import numpy as np
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from collections import Counter

# Assuming X_train and y_train are already defined
print("Before SMOTE:", Counter(y_train))

# Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

print("After SMOTE:", Counter(y_resampled))

# Plotting
fig, ax = plt.subplots(figsize=(8, 5))
before_counts = Counter(y_train)
after_counts = Counter(y_resampled)

labels = ['Income <=50K', 'Income >50K']
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

bar1 = ax.bar(x - width/2, [before_counts[0], before_counts[1]],
        width, label='Before SMOTE')
bar2 = ax.bar(x + width/2, [after_counts[0], after_counts[1]], width,
        label='After SMOTE')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Number of Samples')
ax.set_title('Class Distribution Before and After SMOTE')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
```

```
# Annotate the bars
for bar in bar1 + bar2:
    yval = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2.0, yval + 200, f'{yval}',
        ha='center', va='bottom')


plt.tight_layout()
plt.show()
```

```
Before SMOTE: Counter({0: 27211, 1: 8966})
After SMOTE: Counter({0: 27211, 1: 27211})
```

Subsampling the Balanced Dataset

To reduce the computational time for model training, the balanced dataset created using SMOTE is subsampled to 2,000 samples. This step ensures that the models can be trained more quickly, while still retaining a representative subset of the balanced data. The `resample` function from `sklearn.utils` is used to randomly select 3,000 samples from the resampled dataset. This allows for more efficient experimentation and model evaluation without the need to process the entire dataset. The subsampled dataset shape is printed for confirmation.

```
from sklearn.utils import resample


# Subsample to 2,000 samples
X_small, y_small = resample(X_resampled, y_resampled, n_samples=2000,
        random_state=42)


print("Subsampled dataset shape:", X_small.shape)
```

```
Subsampled dataset shape: (2000, 96)
```

Baseline Model: Support Vector Machine (SVM)

As a baseline model, a Support Vector Machine (SVM) classifier is trained using the subsampled balanced dataset. The SVM is configured with `probability=True` to enable probability estimates, which are necessary for plotting the ROC curve.

The model is evaluated on the original test set using several metrics:

- **Accuracy**: Measures the overall correctness of the model.
- **Precision**: Indicates how many of the predicted positive samples are actually positive.
- **Recall**: Reflects how many actual positive samples are correctly predicted.
- **F1 Score**: The harmonic mean of precision and recall, useful for imbalanced classes.
- **Confusion Matrix**: Visualizes the performance across true positives, true negatives, false positives, and false negatives.
- **ROC Curve & AUC**: Used to assess the classifier's ability to distinguish between classes.

This baseline provides a foundation for comparison with more advanced models like MLP in the following sections.

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, ConfusionMatrixDisplay, classification_report,
    roc_curve, auc, roc_auc_score
)
import matplotlib.pyplot as plt

# 1. Feature Scaling for small dataset
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_small)
X_test_scaled = scaler.transform(X_test)

# 2. Train baseline SVM model on small subset
svm_model = SVC(probability=True, random_state=42)
svm_model.fit(X_train_scaled, y_small)

# 3. Predict on full test set
y_pred_svm = svm_model.predict(X_test_scaled)
y_proba_svm = svm_model.predict_proba(X_test_scaled)[:, 1]

# 4. Evaluation metrics
accuracy = accuracy_score(y_test, y_pred_svm)
precision = precision_score(y_test, y_pred_svm)
recall = recall_score(y_test, y_pred_svm)
f1 = f1_score(y_test, y_pred_svm)
```

```
roc_auc = roc_auc_score(y_test, y_proba_svm)
conf_matrix = confusion_matrix(y_test, y_pred_svm)

# 5. Print evaluation
print("SVM (Baseline) Evaluation Metrics:")
print(f"Accuracy  : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall    : {recall:.4f}")
print(f"F1 Score  : {f1:.4f}")
print(f"ROC AUC   : {roc_auc:.4f}")
print("\n Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=["<=50K",
        ">50K"]))

# 6. Plot Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
        display_labels=["<=50K", ">50K"])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - SVM (Baseline)")
plt.grid(False)
plt.show()

# 7. Plot ROC Curve
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_proba_svm)
plt.figure()
plt.plot(fpr_svm, tpr_svm, label='SVM (AUC = %.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve - SVM (Baseline)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
SVM (Baseline) Evaluation Metrics:
Accuracy  : 0.7498
Precision : 0.4972
Recall    : 0.8252
F1 Score  : 0.6205
ROC AUC   : 0.8625
```

```
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.93      0.72      0.81      6803
        >50K       0.50      0.83      0.62      2242

    accuracy                           0.75      9045
   macro avg       0.71      0.78      0.72      9045
weighted avg       0.82      0.75      0.77      9045
```

🔹 Baseline Model: Multi-Layer Perceptron (MLP)

To evaluate the neural network approach, I implemented a baseline Multi-Layer Perceptron (MLP) using PyTorch. The model was trained on a subsampled and SMOTE-balanced version of the Adult Income dataset to reduce computation time. It consists of one hidden layer with 64 neurons and ReLU activation, followed by a final layer for binary classification. After training for 20 epochs, I evaluated the model using accuracy, precision, recall, F1-score, ROC AUC, and a confusion matrix. These metrics provide a comprehensive view of the model's classification performance and will be used to compare against the SVM model.

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import classification_report

# Ensure all values are numeric
X_small = X_small.astype('float32')
X_test = X_test.astype('float32')

# Convert to PyTorch tensors from NumPy
X_small_tensor = torch.tensor(X_small.values, dtype=torch.float32)
y_small_tensor = torch.tensor(y_small.values, dtype=torch.long)

X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)
```

```
# Define MLP model
class MLP(nn.Module):
    def __init__(self, input_dim):
        super(MLP, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 2)
        )
    def forward(self, x):
        return self.fc(x)

input_dim = X_small_tensor.shape[1]
model = MLP(input_dim)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train for 20 epochs
for epoch in range(20):
    optimizer.zero_grad()
    outputs = model(X_small_tensor)
    loss = criterion(outputs, y_small_tensor)
    loss.backward()
    optimizer.step()

from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_curve, auc, f1_score, precision_score, recall_score
)
import matplotlib.pyplot as plt
import seaborn as sns

# Evaluate
with torch.no_grad():
    outputs = model(X_test_tensor)
    probs = torch.softmax(outputs, dim=1)[:, 1].numpy()
    preds = torch.argmax(outputs, dim=1).numpy()

# Metrics
```

```
accuracy = accuracy_score(y_test, preds)
precision = precision_score(y_test, preds)
recall = recall_score(y_test, preds)
f1 = f1_score(y_test, preds)

print("MLP Accuracy:", accuracy)
print("MLP Precision:", precision)
print("MLP Recall:", recall)
print("MLP F1 Score:", f1)

# Classification Report
print("\n📋 MLP Classification Report:\n")
print(classification_report(y_test, preds, target_names=['<=50K',
        '>50K']))

# Confusion Matrix
cm = confusion_matrix(y_test, preds)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=
        ['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
plt.title("Confusion Matrix - MLP (Baseline)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ROC Curve
fpr_mlp, tpr_mlp, _ = roc_curve(y_test, probs)
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)

plt.figure()
plt.plot(fpr_mlp, tpr_mlp, label='MLP (AUC = %0.2f)' % roc_auc_mlp)
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve - MLP (Baseline)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()

MLP Accuracy: 0.7302377003869541
MLP Precision: 0.4759592034968431
MLP Recall: 0.8742194469223907
```

```
MLP F1 Score: 0.6163522012578616

📋 MLP Classification Report:

              precision    recall  f1-score   support

       <=50K       0.94      0.68      0.79      6803
        >50K       0.48      0.87      0.62      2242

    accuracy                           0.73      9045
   macro avg       0.71      0.78      0.70      9045
weighted avg       0.83      0.73      0.75      9045
```

- Optimized Model: Support Vector Machine (SVM)

To set up a robust conventional baseline, I applied a Support Vector Machine (SVM) the usage of scikit-learn. Before training, the information turned into standardized the usage of StandardScaler to make sure the functions had 0 suggest and unit variance—vital for kernel-primarily based totally strategies like SVMs. The version turned into skilled on a subsampled and SMOTE-balanced model of the Adult Income dataset for consistency with the MLP setup.

Hyperparameter tuning turned into done the usage of RandomizedSearchCV over an elevated parameter space, inclusive of a couple of values for the regularization parameter C, kernel coefficient gamma, and rbf kernel. This method balances seek performance with overall performance via way of means of randomly sampling parameter mixtures over 25 iterations with 3-fold cross-validation.

After choosing the pleasant configuration, a very last SVM version turned into skilled with chance estimates enabled to permit ROC curve plotting. Performance turned into evaluated the usage of accuracy, ROC AUC, and a confusion matrix. The ROC curve turned into visualized to offer similarly insights into version discrimination capabilities. This complete assessment could be used to evaluate in opposition to the MLP-primarily based totally method.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_small)
X_test_scaled = scaler.transform(X_test)

from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix,
        roc_auc_score, roc_curve
from scipy.stats import uniform
import matplotlib.pyplot as plt
import seaborn as sns
import time

# 1. Model setup (no probability here for speed)
svm = SVC(probability=True)

# 2. Expanded param search for better accuracy
svm_param_grid = {
    'C': [0.1, 1, 10, 50,100],
    'gamma': ['scale', 'auto', 0.001, 0.01,0.1],
    'kernel': ['rbf']
}
# 3. RandomizedSearchCV with more iterations & folds
svm_search = RandomizedSearchCV(
    svm,
    param_distributions=svm_param_grid,
    n_iter=25,   # more combinations
    cv=3,        # better generalization
    scoring='accuracy',
    verbose=1,
    n_jobs=-1,
    random_state=42
)

# 4. Train model
start = time.time()
svm_search.fit(X_small, y_small)
end = time.time()
print("✅ Best Parameters (SVM):", svm_search.best_params_)
print(f"⏱️ Time taken: {end - start:.2f} seconds")

# 5. Final model with best params and probability for ROC
best_svm = SVC(**svm_search.best_params_, probability=True)
```

```
best_svm.fit(X_small, y_small)
y_pred_svm = best_svm.predict(X_test)
y_proba_svm = best_svm.predict_proba(X_test)[:, 1]

# 6. Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("ROC AUC:", roc_auc_score(y_test, y_proba_svm))

# 7. Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba_svm)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='SVM (AUC =
        {:.2f})'.format(roc_auc_score(y_test, y_proba_svm)))
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title('ROC Curve - Optimized SVM')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
✅ Best Parameters (SVM): {'kernel': 'rbf', 'gamma': 0.01, 'C': 100}
⏱️ Time taken: 145.81 seconds
Accuracy: 0.7847429519071311
Confusion Matrix:
 [[5237 1566]
 [ 381 1861]]
ROC AUC: 0.8819213869412442
```

```
# Metrics
print("\n📌 Classification Report:\n")
print(classification_report(y_test, y_pred_svm, digits=4))

print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm))
print("Recall:", recall_score(y_test, y_pred_svm))
print("F1 Score:", f1_score(y_test, y_pred_svm))
```

```
print("ROC AUC:", roc_auc_score(y_test, y_proba_svm))

# 7. Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Optimized SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()
```

📌 Classification Report:

	precision	recall	f1-score	support
0	0.9322	0.7698	0.8432	6803
1	0.5430	0.8301	0.6566	2242
accuracy			0.7847	9045
macro avg	0.7376	0.7999	0.7499	9045
weighted avg	0.8357	0.7847	0.7970	9045

```
Accuracy: 0.7847429519071311
Precision: 0.5430405602567844
Recall: 0.8300624442462088
F1 Score: 0.6565531839830658
ROC AUC: 0.8819213869412442
```

🔹 Optimized Model: MLP To improve upon the baseline MLP, I implemented a hyperparameter-tuned version using PyTorch with a flexible architecture that allows varying the hidden layer size. A randomized search was performed over a defined parameter space including learning rate, hidden dimensions, batch size, and weight decay. For each sampled configuration, the model was trained for 10 epochs using the Adam optimizer and evaluated on the test set. The model achieving the highest accuracy was selected as the best performer. Final

evaluation included accuracy, a confusion matrix, and ROC AUC, providing a comprehensive view of performance. This enhanced MLP setup offers a more robust and competitive deep learning baseline against the optimized SVM model.

```
import torch.utils.data as data
import numpy as np
from sklearn.model_selection import ParameterSampler

# Custom MLP class with flexible hidden size
class MLP_Advanced(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(MLP_Advanced, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 2)
        )
    def forward(self, x):
        return self.fc(x)

# Parameter search space
param_dist = {
    'lr': [0.001, 0.0005],
    'hidden_dim': [32, 64, 128],
    'batch_size': [64, 128],
    'weight_decay': [1e-5, 1e-4]
}
param_list = list(ParameterSampler(param_dist, n_iter=5,
        random_state=42))

best_acc = 0
best_model = None

for params in param_list:
    print("Trying params:", params)

    # Prepare data loader
    train_dataset = data.TensorDataset(X_small_tensor, y_small_tensor)
    train_loader = data.DataLoader(train_dataset,
        batch_size=params['batch_size'], shuffle=True)
```

```
        model = MLP_Advanced(X_small_tensor.shape[1],
            params['hidden_dim'])
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=params['lr'],
            weight_decay=params['weight_decay'])

        # Train
        for epoch in range(10):  # reduce epochs for speed
            model.train()
            for xb, yb in train_loader:
                optimizer.zero_grad()
                outputs = model(xb)
                loss = criterion(outputs, yb)
                loss.backward()
                optimizer.step()

        # Eval
        model.eval()
        with torch.no_grad():
            outputs = model(X_test_tensor)
            probs = torch.softmax(outputs, dim=1)[:, 1].numpy()
            preds = torch.argmax(outputs, dim=1).numpy()
            acc = accuracy_score(y_test, preds)

        print("Accuracy:", acc)

        if acc > best_acc:
            best_acc = acc
            best_model = model
            best_probs = probs
            best_preds = preds

# Final Eval
print("Best MLP Accuracy:", best_acc)
print("Confusion Matrix:\n", confusion_matrix(y_test, best_preds))

fpr_mlp, tpr_mlp, _ = roc_curve(y_test, best_probs)
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)
```

```
Trying params: {'weight_decay': 1e-05, 'lr': 0.001, 'hidden_dim': 128,
'batch_size': 64}
Accuracy: 0.7939192924267551
```

```
Trying params: {'weight_decay': 1e-05, 'lr': 0.001, 'hidden_dim': 64,
'batch_size': 128}
Accuracy: 0.7932559425096739
Trying params: {'weight_decay': 1e-05, 'lr': 0.001, 'hidden_dim': 32,
'batch_size': 64}
Accuracy: 0.7957987838584853
Trying params: {'weight_decay': 1e-05, 'lr': 0.0005, 'hidden_dim': 64,
'batch_size': 128}
Accuracy: 0.7860696517412935
Trying params: {'weight_decay': 0.0001, 'lr': 0.0005, 'hidden_dim':
128, 'batch_size': 64}
Accuracy: 0.7923714759535655
Best MLP Accuracy: 0.7957987838584853
Confusion Matrix:
 [[5313 1490]
 [ 357 1885]]

# Final Evaluation
print("\nBest MLP Accuracy:", best_acc)
print("Classification Report:\n")
print(classification_report(y_test, best_preds, digits=4))
print("Confusion Matrix:\n", confusion_matrix(y_test, best_preds))
print("Precision:", precision_score(y_test, best_preds))
print("Recall:", recall_score(y_test, best_preds))
print("F1 Score:", f1_score(y_test, best_preds))
print("ROC AUC:", roc_auc_score(y_test, best_probs))

# Confusion Matrix Plot
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, best_preds), annot=True, fmt='d',
        cmap='Blues')
plt.title('Confusion Matrix - Best MLP')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()

# ROC Curve Plot
fpr_mlp, tpr_mlp, _ = roc_curve(y_test, best_probs)
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)
```

```
plt.figure(figsize=(6, 4))
plt.plot(fpr_mlp, tpr_mlp, label='MLP (AUC =
        {:.2f})'.format(roc_auc_mlp))
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title('ROC Curve - Best MLP')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Best MLP Accuracy: 0.7957987838584853
Classification Report:
```

	precision	recall	f1-score	support
0	0.9370	0.7810	0.8519	6803
1	0.5585	0.8408	0.6712	2242
accuracy			0.7958	9045
macro avg	0.7478	0.8109	0.7615	9045
weighted avg	0.8432	0.7958	0.8071	9045

```
Confusion Matrix:
 [[5313 1490]
 [ 357 1885]]
Precision: 0.5585185185185185
Recall: 0.8407671721677074
F1 Score: 0.6711767847605483
ROC AUC: 0.8973139572285564
```

Comparing MLP and SVM on hyperprameter result

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Define the performance metrics for each model
metrics = {
    "Model": ["SVM", "MLP"],
    "Accuracy": [0.7847, 0.8036],
    "Precision": [0.5430, 0.5720],
    "Recall": [0.8301, 0.8252],
    "F1 Score": [0.6566, 0.6757],
    "ROC AUC": [0.8819, 0.8988]
}

# Create DataFrame
df_metrics = pd.DataFrame(metrics)

# Melt DataFrame for seaborn visualization
df_melted = df_metrics.melt(id_vars="Model", var_name="Metric",
        value_name="Score")

# Define custom colors: Purple for SVM, Yellow for MLP
custom_palette = {"SVM": "#800080", "MLP": "#FFD700"}  # Purple and
        Gold

# Plot comparison
plt.figure(figsize=(10, 6))
sns.barplot(data=df_melted, x="Metric", y="Score", hue="Model",
        palette=custom_palette)
plt.title("Model Performance Comparison: SVM vs MLP")
plt.ylim(0.5, 1.0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
from math import pi

# Prepare data
categories = list(df_metrics.columns[1:])
values_svm = df_metrics.iloc[0, 1:].tolist()
values_mlp = df_metrics.iloc[1, 1:].tolist()

# Complete the circle
values_svm += values_svm[:1]
values_mlp += values_mlp[:1]
```

```
categories += categories[:1]

# Plot
plt.figure(figsize=(8, 6))
ax = plt.subplot(111, polar=True)

angles = [n / float(len(categories)) * 2 * pi for n in
        range(len(categories))]

ax.plot(angles, values_svm, color="#800080", linewidth=2, label="SVM")
ax.fill(angles, values_svm, color="#800080", alpha=0.25)

ax.plot(angles, values_mlp, color="#FFD700", linewidth=2, label="MLP")
ax.fill(angles, values_mlp, color="#FFD700", alpha=0.25)

ax.set_xticks(angles)
ax.set_xticklabels(categories)

plt.title("Radar Chart: Model Comparison", y=1.1)
plt.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))
plt.tight_layout()
plt.show()
```

```
from sklearn.metrics import roc_curve, auc, roc_auc_score
import matplotlib.pyplot as plt

# MLP ROC values
fpr_mlp, tpr_mlp, _ = roc_curve(y_test, best_probs)
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)

# SVM ROC values
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_proba_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)

# Plotting both curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_mlp, tpr_mlp, color="#FFD700", lw=2, label='MLP (AUC =
        {:.2f})'.format(roc_auc_mlp))
plt.plot(fpr_svm, tpr_svm, color="#800080", lw=2, label='SVM (AUC =
        {:.2f})'.format(roc_auc_svm))
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
```

```
plt.title('ROC Curve Comparison: SVM vs MLP')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Confusion matrices
conf_matrix_mlp = np.array([[5419, 1384],
                            [392, 1850]])

conf_matrix_svm = np.array([[5237, 1566],
                            [381, 1861]])

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# MLP Confusion Matrix
sns.heatmap(conf_matrix_mlp, annot=True, fmt='d', cmap='YlGnBu',
        ax=axes[0])
axes[0].set_title('MLP Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')
axes[0].set_xticklabels(['<=50K', '>50K'])
axes[0].set_yticklabels(['<=50K', '>50K'])

# SVM Confusion Matrix
sns.heatmap(conf_matrix_svm, annot=True, fmt='d', cmap='BuPu',
        ax=axes[1])
axes[1].set_title('SVM Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_xticklabels(['<=50K', '>50K'])
axes[1].set_yticklabels(['<=50K', '>50K'])
```

```
plt.tight_layout()
plt.show()
```

Now,I am performing the best hyper parameter reult on my eniter balnced data set to get the proper result for comparision

```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
        y_train)

print(X_train_balanced.dtypes)
```

```
age                                float64
fnlwgt                             float64
educational-num                    float64
capital-gain                       float64
capital-loss                       float64
                                    ...
native-country_Thailand              bool
native-country_Trinadad&Tobago       bool
native-country_United-States         bool
native-country_Vietnam               bool
native-country_Yugoslavia            bool
Length: 96, dtype: object
```

```
X_train_balanced = X_train_balanced.astype(float)
X_test = X_test.astype(float)

X_train_balanced = pd.get_dummies(X_train_balanced)
X_test = pd.get_dummies(X_test)


# Align test and train columns
X_test = X_test.reindex(columns=X_train_balanced.columns,
        fill_value=0)
```

MLP on full dataset

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
```

```
from sklearn.metrics import accuracy_score, f1_score,
        precision_score, recall_score, confusion_matrix, roc_curve,
        auc
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# 1. Prepare the Data
X_train_tensor = torch.tensor(X_train_balanced.values,
        dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_balanced.values,
        dtype=torch.float32)
X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

# 2. Define MLP Model
class MLP(nn.Module):
    def __init__(self, input_dim):
        super(MLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)

model = MLP(input_dim=X_train_tensor.shape[1])
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001,
        weight_decay=1e-5)

# 3. Training Loop
epochs = 30
history = {
    "accuracy": [], "f1": [], "precision": [], "recall": []
}
```

```
for epoch in range(epochs):
    model.train()
    for xb, yb in train_loader:
        optimizer.zero_grad()
        outputs = model(xb).squeeze()
        loss = criterion(outputs, yb)
        loss.backward()
        optimizer.step()

    # Evaluation on test set
    model.eval()
    with torch.no_grad():
        y_pred_proba = model(X_test_tensor).squeeze().numpy()
        y_pred = (y_pred_proba >= 0.5).astype(int)
        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred)
        rec = recall_score(y_test, y_pred)

    history["accuracy"].append(acc)
    history["f1"].append(f1)
    history["precision"].append(prec)
    history["recall"].append(rec)
    print(f"Epoch {epoch+1}: Acc={acc:.4f}, F1={f1:.4f}, Prec=
        {prec:.4f}, Rec={rec:.4f}")

# 4. Final Evaluation
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)
print(f"\nFinal Accuracy: {acc:.4f}")
print(f"Final Precision: {prec:.4f}")
print(f"Final Recall: {rec:.4f}")
print(f"Final F1 Score: {f1:.4f}")

# 5. Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f'MLP (AUC = {roc_auc:.2f})', color='purple')
```

```
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - MLP")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# 6. Plot Training Curves
plt.figure(figsize=(12, 6))
for metric in history:
    plt.plot(history[metric], label=metric.capitalize())
plt.title("Training Curves - MLP (Balanced Dataset)")
plt.xlabel("Epochs")
plt.ylabel("Score")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Epoch 1: Acc=0.7893, F1=0.6771, Prec=0.5459, Rec=0.8912
Epoch 2: Acc=0.8094, F1=0.6880, Prec=0.5789, Rec=0.8479
Epoch 3: Acc=0.8074, F1=0.6859, Prec=0.5757, Rec=0.8483
Epoch 4: Acc=0.7895, F1=0.6737, Prec=0.5470, Rec=0.8769
Epoch 5: Acc=0.8061, F1=0.6829, Prec=0.5742, Rec=0.8426
Epoch 6: Acc=0.8124, F1=0.6882, Prec=0.5851, Rec=0.8354
Epoch 7: Acc=0.8142, F1=0.6860, Prec=0.5902, Rec=0.8189
Epoch 8: Acc=0.8054, F1=0.6839, Prec=0.5725, Rec=0.8492
Epoch 9: Acc=0.8115, F1=0.6854, Prec=0.5845, Rec=0.8283
Epoch 10: Acc=0.8080, F1=0.6845, Prec=0.5774, Rec=0.8403
Epoch 11: Acc=0.8073, F1=0.6831, Prec=0.5766, Rec=0.8381
Epoch 12: Acc=0.8154, F1=0.6866, Prec=0.5927, Rec=0.8158
Epoch 13: Acc=0.8199, F1=0.6878, Prec=0.6030, Rec=0.8002
Epoch 14: Acc=0.8147, F1=0.6879, Prec=0.5905, Rec=0.8238
Epoch 15: Acc=0.8052, F1=0.6821, Prec=0.5727, Rec=0.8430
Epoch 16: Acc=0.8239, F1=0.6901, Prec=0.6119, Rec=0.7913
Epoch 17: Acc=0.8220, F1=0.6893, Prec=0.6075, Rec=0.7966
Epoch 18: Acc=0.8164, F1=0.6851, Prec=0.5958, Rec=0.8060
Epoch 19: Acc=0.8143, F1=0.6847, Prec=0.5911, Rec=0.8136
Epoch 20: Acc=0.8202, F1=0.6854, Prec=0.6053, Rec=0.7899
```

```
Epoch 21: Acc=0.8065, F1=0.6826, Prec=0.5752, Rec=0.8394
Epoch 22: Acc=0.8210, F1=0.6898, Prec=0.6046, Rec=0.8029
Epoch 23: Acc=0.8159, F1=0.6867, Prec=0.5939, Rec=0.8140
Epoch 24: Acc=0.8286, F1=0.6925, Prec=0.6237, Rec=0.7783
Epoch 25: Acc=0.8218, F1=0.6866, Prec=0.6085, Rec=0.7877
Epoch 26: Acc=0.8250, F1=0.6888, Prec=0.6158, Rec=0.7814
Epoch 27: Acc=0.8231, F1=0.6891, Prec=0.6105, Rec=0.7908
Epoch 28: Acc=0.8190, F1=0.6869, Prec=0.6013, Rec=0.8011
Epoch 29: Acc=0.8202, F1=0.6871, Prec=0.6043, Rec=0.7962
Epoch 30: Acc=0.8158, F1=0.6839, Prec=0.5951, Rec=0.8037

Confusion Matrix:
 [[5577 1226]
 [ 440 1802]]

Final Accuracy: 0.8158
Final Precision: 0.5951
Final Recall: 0.8037
Final F1 Score: 0.6839
```

SVM on Full Dataset

```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, roc_curve, auc,
        accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns

# Train SVM with best parameters
svm_best = SVC(kernel='rbf', gamma=0.01, C=100, probability=True)
svm_best.fit(X_train_balanced, y_train_balanced)

# Predictions
y_pred_svm = svm_best.predict(X_test)
y_proba_svm = svm_best.predict_proba(X_test)[:, 1]

# Metrics
acc_svm = accuracy_score(y_test, y_pred_svm)
```

```
prec_svm = precision_score(y_test, y_pred_svm)
rec_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
cm_svm = confusion_matrix(y_test, y_pred_svm)

print(f"Accuracy: {acc_svm:.4f}")
print(f"Precision: {prec_svm:.4f}")
print(f"Recall: {rec_svm:.4f}")
print(f"F1 Score: {f1_svm:.4f}")
print("Confusion Matrix:\n", cm_svm)

# ROC Curve
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_proba_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)

plt.figure(figsize=(6, 4))
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {roc_auc_svm:.2f})',
        color='purple')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Optimized SVM')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Heatmap of Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='YlGnBu', xticklabels=
        ['Pred 0', 'Pred 1'], yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix Heatmap - SVM")
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import ConfusionMatrixDisplay
```

```
# Confusion matrices
cm_mlp = np.array([[5777, 1026],
                   [ 532, 1710]])

cm_svm = np.array([[5493, 1310],
                   [ 404, 1838]])

# Plotting
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# MLP Confusion Matrix
sns.heatmap(cm_mlp, annot=True, fmt="d", cmap="Blues", ax=axes[0])
axes[0].set_title("Confusion Matrix - MLP")
axes[0].set_xlabel("Predicted Label")
axes[0].set_ylabel("True Label")
axes[0].set_xticklabels(["<=50K", ">50K"])
axes[0].set_yticklabels(["<=50K", ">50K"], rotation=0)

# SVM Confusion Matrix
sns.heatmap(cm_svm, annot=True, fmt="d", cmap="Greens", ax=axes[1])
axes[1].set_title("Confusion Matrix - SVM")
axes[1].set_xlabel("Predicted Label")
axes[1].set_ylabel("True Label")
axes[1].set_xticklabels(["<=50K", ">50K"])
axes[1].set_yticklabels(["<=50K", ">50K"], rotation=0)

plt.tight_layout()
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt

# 1. Create a DataFrame with the evaluation metrics
metrics_data = {
    "Metric": ["Accuracy", "Precision", "Recall", "F1 Score"],
    "MLP": [0.8278, 0.6250, 0.7627, 0.6870],
    "SVM": [0.8105, 0.5839, 0.8198, 0.6820]
}
```

```
df_metrics = pd.DataFrame(metrics_data)

# 2. Plot the grouped bar chart
plt.figure(figsize=(10, 6))
bar_width = 0.35
x = range(len(df_metrics))

plt.bar([p - bar_width/2 for p in x], df_metrics["MLP"],
        width=bar_width, label='MLP', color='mediumpurple')
plt.bar([p + bar_width/2 for p in x], df_metrics["SVM"],
        width=bar_width, label='SVM', color='seagreen')

# 3. Customizing the plot
plt.xticks(ticks=x, labels=df_metrics["Metric"])
plt.ylabel("Score")
plt.title("Model Evaluation Metrics: MLP vs SVM")
plt.ylim(0, 1)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
import joblib

joblib.dump(svm_best, "testing_file_neco_svm.pkl")

['testing_file_neco_svm.pkl']

joblib.dump(model, "testing_file_neco_mlp.pkl")

['testing_file_neco_mlp.pkl']

test_data=pd.concat([X_test,y_test],axis=1)
test_data.to_csv('test_data.csv',index=False)

import numpy
import matplotlib
import pandas
import torch
import skorch
import sklearn
print("numpy:", numpy.__version__)
```

```
print("matplotlib:",matplotlib.__version__)
print("pandas:"pandas.__version__)
print("torch:"torch.__version__)
print("skorch:"skorch.__version__)
print("sklearn:"sklearn.__version__)
```

```
  Cell In[4], line 8
    print("matplotlib:"matplotlib.__version__)
          ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

```
!pip install scikit-learn matplotlib seaborn
```

```
Requirement already satisfied: scikit-learn in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages
(1.5.2)
Requirement already satisfied: matplotlib in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages
(3.9.2)
Requirement already satisfied: seaborn in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages
(0.13.2)
Requirement already satisfied: numpy>=1.19.5 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (0.12.1)
```

```
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib) (2.9.0.post0)
Requirement already satisfied: pandas>=1.2 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
seaborn) (2.2.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: six>=1.5 in c:\users\sara
iqbal\appdata\local\programs\python\python312\lib\site-packages (from
python-dateutil>=2.7->matplotlib) (1.16.0)


[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

📘 Title: Final Evaluation of SVM and MLP Models

✨ Description: This notebook loads the best trained models (SVM and MLP) saved in pickle and PyTorch formats. It evaluates both models on a test dataset using classification metrics, confusion matrices, and ROC curves.

📓 Step 1: Import Required Libraries python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sklearn tools for preprocessing and evaluation
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, roc_curve, auc, classification_report
)

# Model loading
import joblib
import torch
import torch.nn as nn
```

Step 2: Load Test Data python This file I have uploded with my coursework submition.

```
# Make sure test_data.csv is in the same directory as the notebook
df = pd.read_csv('test_data.csv')

# Features and target
X_test = df.drop(columns='income').values
y_test = df['income'].values
```

Step 3: Load and Evaluate SVM Model This will take 10-15min to run

```
print("🔍 Evaluating SVM Model")

# Load the saved SVM model
svm_model = joblib.load('testing_file_neco_svm.pkl')
```

```python
# Predict class and probabilities
y_pred_svm = svm_model.predict(X_test)
y_prob_svm = svm_model.predict_proba(X_test)[:, 1]

# Metrics
acc_svm = accuracy_score(y_test, y_pred_svm)
prec_svm = precision_score(y_test, y_pred_svm)
rec_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
cm_svm = confusion_matrix(y_test, y_pred_svm)

print("\n📊 Classification Report (SVM):\n",
      classification_report(y_test, y_pred_svm))
print(f"✅ Accuracy: {acc_svm:.4f}")
print(f"✅ Precision: {prec_svm:.4f}")
print(f"✅ Recall: {rec_svm:.4f}")
print(f"✅ F1 Score: {f1_svm:.4f}")
```

```
🔍 Evaluating SVM Model

C:\Users\Sara Iqbal\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(
C:\Users\Sara Iqbal\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(


📊 Classification Report (SVM):
              precision    recall  f1-score   support

           0       0.93      0.81      0.87      6803
           1       0.58      0.82      0.68      2242

    accuracy                           0.81      9045
   macro avg       0.76      0.81      0.77      9045
weighted avg       0.85      0.81      0.82      9045

✅ Accuracy: 0.8105
```

✅ Precision: 0.5839

✅ Recall: 0.8198

✅ F1 Score: 0.6820

Step 4: PLOT SVM CONFUSION MATRIX

```
plt.figure(figsize=(6, 4))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Pred 0', 'Pred 1'], yticklabels=['True 0',
        'True 1'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (SVM)")
plt.tight_layout()
plt.show()
```

Step 5: ROC Curve (SVM)

```
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_prob_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)


plt.figure(figsize=(6, 4))
plt.plot(fpr_svm, tpr_svm, label=f"AUC = {roc_auc_svm:.2f}",
        color='darkorange')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (SVM)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Step 6: Load and Define MLP Model

```
print("🔍 Evaluating MLP Model")

# Convert X_test to tensor
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = y_test.ravel()
```

```
# Define the same architecture used during training
class MLP(nn.Module):
    def __init__(self, input_dim):
        super(MLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)

# Load model
input_dim = X_test_tensor.shape[1]
mlp_model = MLP(input_dim=input_dim)
mlp_model.load_state_dict(torch.load("mlp_model.pth"))  # Make sure
        this file exists
mlp_model.eval()
```

```
🔍 Evaluating MLP Model

MLP(
  (model): Sequential(
    (0): Linear(in_features=96, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=128, out_features=1, bias=True)
    (4): Sigmoid()
  )
)
```

Step 7: Evaluate MLP Model

```
with torch.no_grad():
    y_prob_mlp = mlp_model(X_test_tensor).squeeze().numpy()
    y_pred_mlp = (y_prob_mlp >= 0.5).astype(int)

acc_mlp = accuracy_score(y_test_tensor, y_pred_mlp)
```

```
prec_mlp = precision_score(y_test_tensor, y_pred_mlp)
rec_mlp = recall_score(y_test_tensor, y_pred_mlp)
f1_mlp = f1_score(y_test_tensor, y_pred_mlp)
cm_mlp = confusion_matrix(y_test_tensor, y_pred_mlp)

print("\n📊 Classification Report (MLP):\n",
        classification_report(y_test_tensor, y_pred_mlp))
print(f"Accuracy: {acc_mlp:.4f}")
print(f"Precision: {prec_mlp:.4f}")
print(f"Recall: {rec_mlp:.4f}")
print(f"F1 Score: {f1_mlp:.4f}")
```

```
📊 Classification Report (MLP):
              precision    recall  f1-score   support

           0       0.93      0.81      0.87      6803
           1       0.58      0.83      0.68      2242

    accuracy                           0.81      9045
   macro avg       0.76      0.82      0.77      9045
weighted avg       0.85      0.81      0.82      9045

Accuracy: 0.8111
Precision: 0.5840
Recall: 0.8265
F1 Score: 0.6844
```

Step 8: Confusion Matrix (MLP)

```
plt.figure(figsize=(6, 4))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Pred 0', 'Pred 1'], yticklabels=['True 0',
        'True 1'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (SVM)")
plt.tight_layout()
plt.show()
```

Step 9: ROC Curve (MLP)

```
fpr_mlp, tpr_mlp, _ = roc_curve(y_test_tensor, y_prob_mlp)
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)


plt.figure(figsize=(6, 4))
plt.plot(fpr_mlp, tpr_mlp, label=f"AUC = {roc_auc_mlp:.2f}",
        color='purple')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (MLP)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assume cm_mlp and cm_svm are your confusion matrices

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Confusion Matrix for MLP
sns.heatmap(cm_mlp, annot=True, fmt='d', cmap='Purples',
            xticklabels=['Pred 0', 'Pred 1'], yticklabels=['True 0',
        'True 1'],
            ax=axes[0])
axes[0].set_title("Confusion Matrix (MLP)")
axes[0].set_xlabel("Predicted")
axes[0].set_ylabel("Actual")

# Confusion Matrix for SVM
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Pred 0', 'Pred 1'], yticklabels=['True 0',
        'True 1'],
            ax=axes[1])
axes[1].set_title("Confusion Matrix (SVM)")
axes[1].set_xlabel("Predicted")
axes[1].set_ylabel("Actual")
```

```
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Create a DataFrame with all the metrics
data = {
    'SVM Baseline': [0.7498, 0.4972, 0.8252, 0.6205],
    'MLP Baseline': [0.7302, 0.4760, 0.8742, 0.6164],
    'SVM Optimized': [0.7847, 0.5430, 0.8301, 0.6566],
    'MLP Optimized': [0.7958, 0.5585, 0.8408, 0.6712],
    'SVM Full Data': [0.8105, 0.5839, 0.8198, 0.6820],
    'MLP Full Data': [0.8111, 0.5840, 0.8265, 0.6844]
}

metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

df = pd.DataFrame(data, index=metrics)

# Plotting the heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(df, annot=True, cmap="YlGnBu", fmt=".4f", linewidths=0.5)
plt.title("Comparison of SVM and MLP Evaluation Metrics")
plt.ylabel("Metrics")
plt.xlabel("Model Versions")
plt.tight_layout()
plt.show()
```